To: P. Muirhead
From: S. Micali
Pages: 13

INTEGRIS SECURITY

Date 2/25/97

Number of pages including cover sheet 12

TO:
Prof. Silvio Micali

Phone
Fax 617 739 3039

FROM: Chini Krishnan
Integris Security, Inc.
333 W. El Camino Real, Suite 270
Sunnyvale, CA 94086

Phone 408 738 4925
Fax Phone 408 738 2159

CC:

REMARKS: ☐ Urgent ☐ For your review ☐ Reply ASAP ☐ Please Comment

Hi Silvio,

Here is an ~~ddN~~ I spec that Paul requested I fax you.

I'm very delighted by the positive conversations you are having with Dr. Hellman & look forward to a Constructive relationship!

Best Regards,
Chini

Feb 25, 97

Hi Don!
Please note that this fax was received today while ~~the date indicated on top~~ is roughly a month earlier!
??? Best,

RECEIVED
FEB 26 1997
F.H. & E. LLP
PATENT DEPT.

Please note that → the activity report of my Xerox 7020 shows Integri's fax as received today

Certificate Validation Trees

DOCUMENT VERSION 1.05

INTEGRIS SECURITY, INC.
1230 OAKMEAD PARKWAY, SUITE 208
SUNNYVALE, CA 94086
TEL: (408) 738-2000
FAX: (408) 738 2159

1. Background

Under the X.509 directory infrastructure, certificate Revocation Lists (CRLs) are issued by Certificate Authorities (CAs) to revoke certificates. Because revoked certificates cannot be trusted, it is essential that systems using certificates include secure mechanisms to verify revocation status. Without such verification, a few compromised certificates could be used for wide-scale fraud. There will soon be millions of certificates accepted by protocols such as SSL and S/MIME and some fraction of these will need to be revoked.

Protocols should be optimized for the case where certificates are not revoked, since in practice revoked certificates will only be encountered in extraordinary circumstances (i.e., cases of attempted fraud). CRLs are extremely inefficient, since verification of a certificate's status requires the entire CRL. As a result, CRLs will become unacceptably large in large-scale systems involving millions of certificates. Furthermore, if certificate chaining and attribute certificates are used, users must obtain separate CRLs from each CA in the chain and every attribute certificate issuer. If CRLs are used for revocation, systems must be able to get CRLs from every CA. (This problem can be alleviated somewhat by designating CRL repositories, but these systems would consume an enormous amount of network bandwidth.) Complete new CRLs must be downloaded regularly since users cannot reliably determine whether a given certificate is valid without the complete CRL. Network traffic dedicated to CRL distribution could potentially exceed all other traffic combined, especially if CRLs are updated frequently.

While a variety of measures, such use of only very short-term certificates, can alleviate the situation somewhat, certificate validation trees eliminate the major problems associated with CRLs without sacrificing security. This document defines structures for certificate revocation trees.

2. System Overview

Certificate Validation Trees (CVT) trees provide an efficient way for certificate acceptors to determine whether certificates have been revoked and allow certificate holders to demonstrate that their own certificates have not been revoked. Trees are computed in advance by a "semi-trusted" party (such as Integris Security or a CA), which processes the data from one or more CRLs into tree form. The issuer is described as "semi-trusted," since the operation of the tree issuer is publicly auditable. Any independent party can verify that the tree issuer has not inappropriately revoked valid certificates or omitted revoked certificates from the tree. The tree's root node is then digitally signed.

To prove that one or more certificates have not been revoked, the tree leaves spanning the certificates in question are required, along with the supporting hashes binding the leaves to the tree's root and the digitally-signed root.

Certificate validation involves four kinds of entities: CAs, CVT issuers, confirmation issuers, and verifiers. The CAs are responsible for issuing certificates and issuing CRLs. A CVT issuer collects and verifies CRLs from one or more CAs then constructs a digitally-signed, dated certificate validation tree. The signed tree structure is then provided to confirmation issuers, which respond to users' requests for revocation information regarding specific certificates.

This document defines formats for the confirmation request, confirmation response, and CVT.

3. Data Formats

A **Confirmation Request** is passed from a verifier to a confirmation issuer and identifies one or more certificates whose revocation status is to be determined.

```
ConfirmationRequest     ::=   SEQUENCE {
      version                 Version DEFAULT v1,
      applicationID           INTEGER,
      hash                    AlgorithmIdentifier,
      requestList             SEQUENCE OF Request,
      requestExtensions       Extensions OPTIONAL }

Request                 ::=   SEQUENCE {
      issuerNameAndKeyHash    Hash,
      serialNumber            CertificateSerialNumber }

IssuerNameAndKey        ::=   SEQUENCE {
      issuer                  Name,
      issuerPublicKey         OCTET STRING } --Use what format!!!?--
```

The **issuerNameAndKeyHash** is computed by hashing an **IssuerNameAndKey** field constructed for the CA in question using a cryptographic hash function (i.e., SHA-1).

The confirmation issuer responds with a **ConfirmationResponse** structure:

```
ConfirmationResponse    ::=   SEQUENCE {
      resultStatus                  ResultStatus,
      certificateConfirmations      SEQUENCE OF
                                        CertificateConfirmations,
      forwardingInfo          [1]   ForwardingInfo OPTIONAL
            -- Should make forwardingInfo an extension? !!! --
            -- NEED TO ADD [OPTIONAL] EXTENSIONS HERE--
}
```

```
ResultStatus            ::=   ENUMERATED {
      successful                (0),   --Response has valid confirmations--
      someCertsUnknown          (1),   --When is this returned?--
      malformedRequest          (2),   --Illegal confirmation request--
      requestorUnauthorized     (3),   --User not authorized to use issuer--
      internalError             (4),   --Internal error in issuer--
      serverHasNoTree           (5),   --Issuer tree missing or out of date--
      noGlobalTree              (6)    --Request requires a global tree--
}

ForwardingInfo                  ::=   SEQUENCE {
      siteNameList                    SEQUENCE OF OCTET STRING }

CertificateConfirmations        ::=   SEQUENCE {
      treeHeader                      SIGNED { SEQUENCE {
            version                   Version DEFAULT v1,
            minVersionToRead          Version DEFAULT v1,
            signature                 AlgorithmIdentifier,
            issuer                    GeneralNames,
            thisUpdate                GeneralizedTime,
            nextUpdate                GeneralizedTime,
            validUntil                GeneralizedTime,
            hash                      AlgorithmIdentifier,
            totalLeafCount            LeafCount,
            rootHash                  Hash,
            crtExtensions             Extensions OPTIONAL }},
      treeLeafAndPath                 SEQUENCE SIZE (1..MAX) OF
                                            TreeLeafAndPath }

Version                 ::=   INTEGER { v1(0) }

TreeSerialNumber        ::=   INTEGER

LeafCount               ::=   INTEGER (1..MAX)

Hash                    ::=   OCTET STRING
                              -- Length must equal hash size --

TreeLeafAndPath         ::=   SEQUENCE {
      treeLeaf                TreeLeaf,
      leafPath                SEQUENCE OF Hash }

-- TreeLeaf must be DER encoded. --
TreeLeaf                ::=   SEQUENCE {
      leafPosition            LeafPosition,
```

```
        leafData                LeafData }

LeafData                ::=     CHOICE {
        unknownCA_Range         [0]     UnknownCARange,
        leafRange               [1]     LeafRange }

UnknownCARange          ::=     SEQUENCE {
        issuerPublicKeyHash1    Hash,
        issuerPublicKeyHash2    Hash }

LeafRange               ::=     SEQUENCE {
        issuerPublicKeyHash     [0]     Hash OPTIONAL,
        thisCrlUpdate           [1]     GeneralizedTime OPTIONAL,
        nextCrlUpdate           [2]     GeneralizedTime OPTIONAL,
        validUntil              [3]     GeneralizedTime OPTIONAL,
        revocationDate          [4]     GeneralizedTime OPTIONAL,
        rangeMinStatus                  RevocationStatus,
        rangeStatus                     RevocationStatus,
        rangeMinimum            [5]     CertSerialNumber OPTIONAL,
        rangeMaximum            [6]     CertSerialNumber OPTIONAL,
        leafExtensions          [7]     Extensions OPTIONAL }

LeafPosition            ::=     INTEGER

RevocationStatus        ::=     ENUMERATED {
        revokedReasonUnspecified ( 0 ),
        keyCompromise           ( 1 ),
        cACompromise            ( 2 ),
        affiliationChanged      ( 3 ),
        superseded              ( 4 ),
        cessationOfOperation    ( 5 ),
        certificateHold         ( 6 ),
        expiredNormally         ( 7 ),
        noAcceptableCRL         ( 8 ),
        unsupportedRange        ( 32 ),
        unknownStatusOkay       ( 33 ),
        validCertificate        ( 64 ) }

CertSerialNumber ::= INTEGER
```

The treeHeader is digitally signed by the tree issuer and specifies the version of the current certificate validation tree, the oldest version with which the structure is backward-compatible, the tree serial number, the algorithm used to sign the header, the tree issuer's public key (hashed), the time of the tree's issuance, optionally the time of the next tree's

issuance, optionally the time when the tree expires, the secure hash function used to construct the tree, the total number of leaves present in the tree, and finally the tree's root hash. The tree serial number values should be assigned sequentially by the tree issuer, beginning with zero The tree issuer can either be a trusted third party entity (i.e., Integris Security) or the CA who originally issued the revoked certificates in the tree.

4. Verifying confirmations

[!!! This section has not been polished yet]

Every **TreeLeafAndPath** is a cryptographic assertion from the revocation tree issuer that no known revoked certificates exist within some range. Either the certificate in question is not known to be revoked, is revoked, is from a revoked CA, or is from a CA for which no CRL is present in the tree. (The third result is important, since applications might accept certificates from CAs whose CRLs are not in the tree, but no application should accept revoked certificates.)

To determine a certificate's status given a treeHeader and treeLeafandPath, do the following:

1. Check version and signature algorithm in treeHeader.
2. Verify that the issuer Name in the treeHeader corresponds to a trusted party or the CA who issued the certificate in question. If the name is unrecognized, the confirmation is invalid.
3. Check that the tree is acceptably recent by checking the header validUntil field. (See step 9 for checking of the leaf's time fields.)
4. Check that the Hash algorithm is acceptable (i.e., SHA).
5. Verify the signature on the treeHeader using the public key corresponding to the issuer's name.
6. Verify that leafPosition < totalLeafCount.
7. Using the leafPosition, verify that the leafPath binds the hash of the leaf to the rootHash in the signed treeHeader.
8. If the LeafData is of type UnknownCA_Range, verify that the hash of the CA's public key lies between issuerPublicKeyHash1 and issuerPublicKeyHash2. If so, the CA is not in the tree. If not, the leaf is inappropriate.
9. If the LeafData is of type LeafRange:
    - Verify that the issuerPublicKeyHash matches the hash of the CA's public key. (If issuerPublicKeyHash is omitted, it is identical to the tree issuer.) If the CA does not match, the leaf is inappropriate.
    - If present, verify that thisCrlUpdate, nextCrlUpdate, and validUntil are acceptable.
    - Verify that rangeMinimum, if present, is smaller than or equal to the serial number of the certificate in question. Also verify that rangeMaximum, if present, is larger than the serial number of the certificate in question.

- Check the RevocationStatus field. If 32 ≤ RevocationStatus < 64, the certificate's status is unknown. The value 32 corresponds to an unknown range (i.e., absolutely no revocation information is available). The value 33 indicates that no current revocation information is present for this range, but this is normal so applications may decide to accept certificates in the range.
- If 64 ≤ RevocationStatus < 96 or if certificateSerialNumber ≠ rangeMinimum, the certificate is valid. Otherwise the certificate is revoked for the reason specified in RevocationStatus.

The **issuerPublicKeyHash** values are computed as the hash of the DER-encoded public key of the CA. The time in **thisCrlUpdate** and optionally **nextCrlUpdate** in the LeafRange structure equal **thisUpdate** and **nextUpdate** from the CRL from the specified CA which is used in the tree.

The recipient of a **treeLeafAndPath** can use the **leafPath** to verify that each **treeLeaf** is cryptographically bound to the root:

Let **maxDepth** = $\lceil \log_2(\textbf{totalLeafCount}) \rceil$ -- $\lceil n \rceil$ *rounds up if not an integer.* --
Let y = **leafPosition**.
Let i = 0.
hash[0] = **HASHED** { **treeLeaf** }.
For **x** = 0 upto **maxDepth**-1
    Let y' = (y ⊕ 2^x) - ((y ⊕ 2^x) mod 2^x).
    if y' < y then
        hash[x+1] = **HASHED** { **hash**[x] | **leafPath**[x-i] }.
    else if y' < **totalLeafCount** then
        hash[x+1] = **HASHED** { **leafPath**[x-i] | **hash**[x] }.
    else
        i = i + 1.
EndFor.
Assert (**hash**[**maxDepth**] = **rootHash**).

The operation of the revocation tree issuer can be verified easily, since anyone can verify that

1) all known revoked certificates and CAs should be contained in the tree, and
2) only known revoked certificates and CAs are contained in the tree.

A tree can include revocations for multiple CAs so that a single CertificateConfirmations structure can provide assurances for all certificates in a chain. The tree can even include revoked non-X.509 certificates (in which case the structure of TreeLeaf might change).

CertificateConfirmations messages are quite small; even if the tree includes many millions of revoked certificates. For example, each leafPath from a tree with a billion leaves using a

hash size of 20 bytes would require about 600 bytes. A single asymmetric signature verification can provide assurances for all certificates in the chain.

Tree construction can be performed in linear time. Using the tree and signed root, confirmations can be constructed very efficiently using only insecure hardware.

5. ASN.1 Module

```
CertRevocationTreesDefinitions { joint-iso-ccitt (2) country (16) us (840)
organization (1) integris (-TBD!!!-) modules (1)
certRevocationTreesDefinitions (1) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS all definitions, in particular:
--      CertificateConfirmations,
--      -- Message syntax to communicate certificate confirmations --
--      ConfirmationRequest
--      -- Message syntax to request certificate confirmations --

IMPORTS
        AlgorithmIdentifier, SIGNED, Extensions
                FROM AuthenticationFramework {joint-iso-ccitt ds(5)
                module(1) authenticationFramework(7) 2}
                -- Note definition of Extensions requires application of
                -- Technical Corrigendum 1 --
        GeneralNames
                FROM CertificateExtensions {joint-iso-ccitt ds(5)
                module(1) certificateExtensions(26) 0} ;
-- Message syntax to request certificate confirmations --

ConfirmationRequest     ::=     SEQUENCE {
        version                 Version DEFAULT v1,
        applicationID           INTEGER,
        hash                    AlgorithmIdentifier,
        requestList             SEQUENCE OF Request,
        requestExtensions       Extensions OPTIONAL }


Request                 ::=     SEQUENCE {
        issuerNameAndKeyHash    Hash,
        serialNumber            CertSerialNumber }
```

```
-- Message syntax to communicate certificate confirmations --
ConfirmationResponse      ::=   SEQUENCE {
      resultStatus                    ResultStatus,
      certificateConfirmations        SEQUENCE OF
                                          CertificateConfirmations,
      forwardingInfo            [1]   ForwardingInfo OPTIONAL
}

ResultStatus              ::=   ENUMERATED {
      successful                  ( 0 ),
      someCertsUnknown            ( 1 ),
      malformedRequest            ( 2 ),
      requestorUnauthorized       ( 3 ),
      internalError               ( 4 ),
      serverHasNoTree             ( 5 ),
      noGlobalTree                ( 6 ) }


ForwardingInfo                    ::=   SEQUENCE {
      siteNameList                SEQUENCE OF OCTET STRING }

CertificateConfirmations          ::=   SEQUENCE {
      treeHeader                  SIGNED { SEQUENCE {
            version               Version DEFAULT v1,
            minVersionToRead      Version DEFAULT v1,
            signature             AlgorithmIdentifier,
            issuer                GeneralNames,
            thisUpdate            GeneralizedTime,
            nextUpdate            GeneralizedTime,
            validUntil            GeneralizedTime,
            hash                  AlgorithmIdentifier,
            totalLeafCount        LeafCount,
            rootHash              Hash,
            crtExtensions         Extensions OPTIONAL }},
      treeLeafAndPath             SEQUENCE SIZE (1..MAX) OF
                                        TreeLeafAndPath }

Version                   ::=   INTEGER { v1(0) }

TreeSerialNumber          ::=   INTEGER

LeafCount                 ::=   INTEGER (1..MAX)

Hash                      ::=   OCTET STRING
                                -- Length must equal hash size --
```

```
TreeLeafAndPath          ::=   SEQUENCE {
        treeLeaf                   TreeLeaf,
        leafPath                   SEQUENCE OF Hash }

-- TreeLeaf must be DER encoded --
TreeLeaf                 ::=   SEQUENCE {
        leafPosition               LeafPosition,
        leafData                   LeafData }

LeafData                 ::=   CHOICE {
        unknownCA_Range    [0]     UnknownCARange,
        leafRange          [1]     LeafRange }

UnknownCARange           ::=   SEQUENCE {
        issuerPublicKeyHash1       Hash,
        issuerPublicKeyHash2       Hash }

LeafRange                ::=   SEQUENCE {
        issuerPublicKeyHash [0]    Hash OPTIONAL,
        thisCrlUpdate       [1]    GeneralizedTime OPTIONAL,
        nextCrlUpdate       [2]    GeneralizedTime OPTIONAL,
        validUntil          [3]    GeneralizedTime OPTIONAL,
        revocationDate      [4]    GeneralizedTime OPTIONAL,
        rangeMinStatus             RevocationStatus,
        rangeStatus                RevocationStatus,
        rangeMinimum        [5]    CertSerialNumber OPTIONAL,
        rangeMaximum        [6]    CertSerialNumber OPTIONAL,
        leafExtensions      [7]    Extensions OPTIONAL }

LeafPosition             ::=   INTEGER

RevocationStatus         ::=   ENUMERATED {
        unspecified                ( 0 ),
        keyCompromise              ( 1 ),
        cACompromise               ( 2 ),
        affiliationChanged         ( 3 ),
        superseded                 ( 4 ),
        cessationOfOperation       ( 5 ),
        certificateHold            ( 6 ),
        expiredNormally            ( 7 ),
        noAcceptableCRL            ( 8 ),
        unsupportedRange           ( 32 ),
        unknownStatusOkay          ( 33 ),
        validCertificate           ( 64 ) }
```

CertSerialNumber ::= INTEGER

END

[19]

[11] 4,309,569

[45] Jan. 5, 1982

References Cited

PATENT DOCUMENTS

Hellman et al. 375/2

Diffie et al. Test.